Homework for Computer Architecture

Fhaheem Tadamarry

Spring 2025

Homework 1

1. **Amdahl's Law - Solve Problem 1.15 (a,b,c,d) on pages 74-75.**
The formula for Amdahl's Law is:

$$speedup = \frac{1}{1-F+\frac{F}{N}}$$

- F = fraction parallelizable
- N = number of cores

(a) **How much speedup would result from running application A on the entire 22-core processor, as compared to running it serially?**
Application A would have a speedup of:

$$speedup = \frac{1}{1-0.50+\frac{0.50}{22}} = 1.91$$

Therefore, the speedup would be 1.91 times faster than running it serially.

(b) **How much speedup would result from running application D on the entire 22-core processor, as compared to running it serially?**
Application D would have a speedup of:

$$speedup = \frac{1}{1-0.90+\frac{0.90}{22}} = 9.00$$

Therefore, the speedup would be 9.00 times faster than running it serially.

(c) **Given that application A requires 41% of the resources, if we statistically assign it 41% of the cores, what is the overall speedup if A is run parallelized but everything else is run serially?**
Application A requires 41% of the resources so it would get 41% of the 22 cores, $(22 * \frac{41}{100})$ is 9 cores. Using the formula:

$$speedup = \frac{1}{1-0.50+\frac{0.50}{9}} = 1.80$$

Therefore, the overall speedup would be 1.80 times faster than running it serially.

(d) **What is the overall speedup if all four applications are statically assigned some of the cores, relative to their percentage of resource needs, and all run parallelized?** The number of cores assigned to each application is directly porportional to the percentage of resources needed.

Application	% Resources Needed	% Parallelizable	Cores Assigned
A	41%	50	$22 \times 0.41 = 9$
B	27%	80	$22 \times 0.27 = 6$
C	18%	60	$22 \times 0.18 = 4$
D	14%	90	$22 \times 0.14 = 3$

Using the formula:

$$speedup_A = \frac{1}{1-0.50+\frac{0.50}{9}} = 1.80 \qquad speedup_C = \frac{1}{1-0.60+\frac{0.60}{4}} = 1.82$$

$$speedup_B = \frac{1}{1-0.80+\frac{0.80}{6}} = 3.00 \qquad speedup_D = \frac{1}{1-0.90+\frac{0.90}{3}} = 2.50$$

$$speedup_{overall} = (0.41\times 1.80)+(0.27\times 3.00)+(0.18\times 1.82)+(0.14\times 2.50) = 2.25$$

(e) **Given acceleration through parallelization, what new percentage of the resources are the applications receiving, considering only active time on their statically-assigned cores?**
The applications run faster, they use less total execution time which would change their effective resource usage.

Application	Old %	Speedup	New %
A	41%	1.80	$\frac{41}{1.80} = 22.8\%$
B	27%	3.00	$\frac{27}{3.00} = 9.0\%$
C	18%	1.82	$\frac{18}{1.82} = 9.9\%$
D	14%	2.50	$\frac{14}{2.50} = 5.6\%$

2. **Decide whether each of the following is true or false. Add brief explanation (1-2 sentences) to get full credit.**

(a) **The performance of the system is limited by the fastest component even if some components are made $7\times$ slower**
Answer: False
Performance is limited by the slowest component, not the fastest component. If some components are made slower, the overall performance will be limited by the slowest component.

(b) **You can pay attention to Amdahl's law because it is still applicable**
Answer: True
It is still applicable because it is a general principle that applies to all parallel systems. It defines the fundamental limits of speedup when improving only a portion of execution. As core counts increase, the law explains why the serial portion of a workload prevents unlimited performance scaling.

(c) **CPI rating of a processor is a good metric to measure its performance.**
Answer: False
CPI (Cycles Per Instruction) is not a good metric to measure performance because it does not take into account the clock speed of the processor. A processor with a lower CPI may still have lower performance if it has a lower clock speed.

(d) **The future of performance improvement will mostly be dependent on parallelization of programming rather than blindly adding multiple cores to a chip.**
Answer: True
Only increasing the number of cores does not guarantee performance improvement. The future of performance improvement will mostly be dependent on parallelization of programming to take advantage

of multiple cores. This is shown by Amdahl's Law, which states that the speedup of a program is limited by the serial portion of the program.

3. **Fabrication Cost: Solve Problem 1.2 of `Case Study 1` on page 68. Assume Wafer Yield is 100%.** *(Hint: Review Textbook Examples on pages 33-34.)*
The formula used for Die Yeild is:

$$Die\ Yeild = \frac{Wafer\ Yeild}{[1 + (Defect\ Density \times Die\ Area)]^N}$$

We also are given:

- Wafer Diameter = 450 mm

(a) **How much profit do you make on each wafter of Phoenix[8] chips?**
We first need to compute the Die Yeild:

$$Die\ Yeild = \frac{1}{[1 + (0.04 \times \frac{200}{100})]^{14}} = 0.325$$

Next, we compute the number of dies per wafer:

$$Dies\ per\ Wafer = \frac{Wafer\ Area}{Die\ Area} = \frac{\pi \times (\frac{450}{2})^2}{200} = 795\ Dies$$

Where the good dies per wafer is:

$$Good\ Dies\ per\ Wafer = Dies\ per\ Wafer \times Die\ Yeild = 795 \times 0.325 = 259\ Dies$$

Finally we compute the profit per wafer:

$$Profit\ per\ Wafer = Good\ Dies\ per\ Wafer \times Profit\ per\ Die = 259 \times \$30 = \$7,700$$

Therefore, the profit per wafer of Phoenix[8] chips is $7,700.

(b) **How much profit do you make on each wafer of RedDragon chips?**
We first need to compute the Die Yeild:

$$Die\ Yeild = \frac{1}{[1 + (0.04 \times \frac{120}{100})]^{14}} = 0.525$$

Next, we compute the number of dies per wafer:

$$Dies\ per\ Wafer = \frac{Wafer\ Area}{Die\ Area} = \frac{\pi \times (\frac{450}{2})^2}{120} = 1325\ Dies$$

Where the good dies per wafer is:

$$Good\ Dies\ per\ Wafer = Dies\ per\ Wafer \times Die\ Yeild = 1325 \times 0.525 = 695\ Dies$$

Finally we compute the profit per wafer:

$$Profit\ per\ Wafer = Good\ Dies\ per\ Wafer \times Profit\ per\ Die = 695 \times \$15 = \$10,440$$

Therefore, the profit per wafer of RedDragon chips is $10,440.

(c) **If your deamnd is 50,000 RedDragon chips per month and 25,000 Phoenix^8 chips per month, and your facility can fabricate 70 wafers a month, how many wafers should you make of each chip?**
We first need to compute how many total wafers are needed for each chip:

$$Wafers\ for\ RedDragon = \frac{50,000}{695} = 72\ Wafers$$

$$Wafers\ for\ Phoenix^8 = \frac{25,000}{259} = 97\ Wafers$$

The total number of wafers needed is:

$$Total\ Wafers = 72 + 97 = 169\ Wafers$$

We would need 169 Wafers, however the facility can fabricate 70 wafers a month, therefore we have to strategically allocate them based on demand.
Next, we calculate the proportional demand for each chip:

$$RedDragon\ Demand = \frac{72}{169} = 42.6\%$$

$$Phoenix^8\ Demand = \frac{97}{169} = 57.4\%$$

Therefore, we should make $70 \times 0.426 = 30$ wafers of RedDragon chips and $70 \times 0.574 = 40$ wafers of Phoenix^8 chips.

4. **A cell phone performs very different tasks, including streaming music, streaming video, and reading email. These tasks perform very different computing tasks. Battery life and overheating are two common problems for cell phones, so reducing power and energy consumption are critical. In this problem, we consider what to do when the user is not using the phone to its full computing capacity. For these problems, we will evaluate an unrealistic scenario in which the cell phone has no specialized processing units. Instead, it has a quad-core, general purpose processing unit. Each core uses 1 W at full use. For email-related tasks, the quad-core is $8\times$ as fast as necessary**
999p8089 *(Hint: Review Textbook Example on page 25.)*

We will need the following equations for this problem:

$$Dynamic\ Power = \frac{1}{2} \times Capacitance\ Load \times Voltage^2 \times Frequency$$

$$Dynamic\ Energy = Dynamic\ Power \times Time$$

(a) **How much dynamic energy and power are required compared to running at full power? First, suppose that the quad-core operates for $\frac{1}{2}$ of the time and is idle for the rest of the time. Compare total dynamic energy as well as dynamic power while the core is running.**
At full utilization, the total power is:

$$P_{\text{full}} = 4 \times 1W = 4W$$

Since the processor operates for only half the time, the total energy consumption is:

$$E_{\text{half}} = P_{\text{full}} \times 0.5 = 4W \times 0.5s = 2J$$

The average power over the full period:

$$P_{\text{avg}} = \frac{E_{\text{half}}}{1s} = 2W$$

Thus, the **total energy used is 2J**, while the **average power consumption is 2W**.

(b) **How much dynamic energy and power are required using frequency and voltage scaling? Assume frequency and voltage are both reduced to $\frac{1}{4}$ the entire time.**
Since **energy is proportional to** V^2:

$$\frac{E_{\text{new}}}{E_{\text{old}}} = \left(\frac{1}{4}\right)^2 = \frac{1}{16}$$

Since **full energy was 4J**, the new energy is:

$$E_{\text{scaled}} = 4J \times \frac{1}{16} = 0.25J$$

Power scales as:

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \left(\frac{1}{4}\right)^2 \times \left(\frac{1}{4}\right) = \frac{1}{64}$$

Since **full power was 4W**, the new power is:

$$P_{\text{scaled}} = 4W \times \frac{1}{64} = 0.0625W$$

Thus, the **total energy used is 0.25J**, while the **power consumption is 0.0625W**.

(c) **Now assume the voltage may not decrease below 70% of the original voltage. This voltage is referred to as the voltage floor, and any voltage lower than that will lose the state. Therefore, while the frequency can keep decreasing, the voltage cannot. What are the dynamic energy and power savings in this case?**

Since **energy is proportional to** V^2:

$$\frac{E_{\text{new}}}{E_{\text{old}}} = (0.7)^2 = 0.49$$

Since **full energy was 4J**, the new energy is:

$$E_{\text{floor}} = 4J \times 0.49 = 1.96J$$

Power scales as:

$$\frac{P_{\text{new}}}{P_{\text{old}}} = (0.7)^2 \times \left(\frac{1}{4}\right) = 0.49 \times 0.25 = 0.1225$$

Since **full power was 4W**, the new power is:

$$P_{\text{floor}} = 4W \times 0.1225 = 0.49W$$

Thus, the **total energy used is 1.96J**, while the **power consumption is 0.49W**.

(d) **How much energy is used with a dark silicon approach? This involves creating specialized ASIC hardware for each major task and power gating those elements when not in use. Only one general-purpose core would be provided, and the rest of the chip would be filled with specialized units. For email, the one core would operate for 25% the time and be turned completely off with power gating for the other 75% of the time. During the other 75% of the time, a specialized ASIC unit that requires 20% of the energy of a core would be running.**

Step 1: Compute Power

$$P_{\text{core}} = 1W$$

$$P_{\text{core avg}} = 1W \times 0.25 = 0.25W$$

Step 2: Compute Power for ASIC

$$P_{\text{ASIC}} = 0.2W$$

$$P_{\text{ASIC avg}} = 0.2W \times 0.75 = 0.15W$$

Step 3: Compute Total Energy

$$E_{\text{dark}} = (0.25W + 0.15W) \times 1s = 0.4J$$

Thus, the **total energy used is 0.4J**, while the **power consumption is 0.4W**.

(a) **How much dynamic energy and power are required compared to running at full power? First, suppose that the quad-core operates for $\frac{1}{2}$ of the time and is idle for the rest of the time. Compare total dynamic energy as well as dynamic power while the core is running.**

(b) How much dynamic energy and power are required using frequency and voltage scaling. Assume frequency and voltage are both reduced to $\frac{1}{4}$ the entire time.

Answer:
(Provide your answer here)

(c) Now assume the voltage may not decrease below 70% of the original voltage. This voltage is referred to as the voltage floor, and any voltage lower than that will lose the state. Therefore, while the frequency can keep decreasing, the voltage cannot. What are the dynamic energy and power savings in this case?

Answer:
(Provide your answer here)

(d) How much energy is used with a dark silicon approach? This involves creating specialized ASIC hardware for each major task and power gating those elements when not in use.

Answer:
(Provide your answer here)

5. **Effective CPI: Solve Problem A.3 in Appendix A on page A-48.**

   Answer:
   (Provide your answer here)

6. **CISC Architecture: Pick any CISC architecture and examine its instruction set architecture and answer the following questions. You can search online for the ISA summary for the processor you have chosen. Cite the source(s) you have used for this problem.**

   Answer:
   (Provide your answer here)

Homework 2

1. (6 pts) Briefly describe six basic cache optimization techniques.

   (a) **Larger Block Size:** Increasing the block size reduces compulsory misses by exploiting spatial locality, though excessively large blocks may increase conflict and capacity misses.

   (b) **Higher Associativity:** Increasing associativity helps to reduce conflict misses but increases access time and complexity.

   (c) **Multilevel Caching:** Using multiple levels of cache helps balance fast access time with lower miss rates by leveraging a smaller L1 and larger L2/L3 caches.

   (d) **Critical Word First/Early Restart:** These techniques prioritize the word needed by the CPU, reducing stall time by allowing partial block utilization before the full cache block is loaded.

   (e) **Way Prediction:** Reduces power consumption and hit time by predicting the correct way in set-associative caches and accessing only the predicted way.

   (f) **Compiler Optimizations:** Techniques such as loop interchange and blocking optimize memory access patterns to reduce cache misses.

2. (8 pts) Briefly describe eight advanced cache optimization techniques.

   (a) **Hardware Prefetching:** The hardware anticipates future memory accesses and preloads data into the cache.

   (b) **Compiler Prefetching:** The compiler inserts prefetch instructions to load data before it is needed.

   (c) **Victim Cache:** A small fully associative cache that stores recently evicted blocks to reduce conflict misses.

   (d) **Non-blocking Caches:** Allows other memory accesses to continue while waiting for a cache miss to be serviced.

   (e) **Write Buffering and Merging:** A write buffer temporarily holds write operations to reduce stalls, and merging allows multiple writes to the same address to be consolidated.

   (f) **Adaptive Replacement Policies:** Adjusts cache replacement strategies dynamically to optimize performance.

   (g) **Sector Caches:** Divides large blocks into sectors, reducing unnecessary data transfers.

   (h) **Trace Caches:** Stores sequences of executed instructions to improve branch prediction.

3. (10 pts) Solve problem B.1 on page B-60.

   (a) Given a cache with a block size of 64 bytes and a miss penalty of 50 cycles, the average memory access time (AMAT) can be calculated as:

$$AMAT = HitTime + (MissRate \times MissPenalty) \tag{1}$$

   Assume a hit time of 2 cycles and a miss rate of 5%, then:

$$AMAT = 2 + (0.05 \times 50) = 4.5 \text{ cycles} \tag{2}$$

4. (10 pts) Solve problem B.2 on page B-60.

   (a) In a fully associative cache with 4-way associativity and a total of 1024 blocks, the number of sets is:

$$\text{Number of sets} = \frac{\text{Total blocks}}{\text{Associativity}} = \frac{1024}{4} = 256 \tag{3}$$

5. (10 pts) Solve problem B.5 on page B-63.

   (a) Consider a cache with a hit time of 1 cycle and a miss penalty of 100 cycles, if the miss rate is 2%, the AMAT is:

$$AMAT = 1 + (0.02 \times 100) = 3 \text{ cycles} \tag{4}$$

6. (10 pts) Solve problem B.12 on page B-65.

   (a) A large cache reduces miss rate but increases access time. Given a 4MB cache with an access time of 10 cycles and a 512KB cache with an access time of 4 cycles, if the miss rate for the larger cache is 1% and for the smaller cache is 5%, then AMAT for each is:

$$AMAT_{4MB} = 10 + (0.01 \times 100) = 11 \text{ cycles} \tag{5}$$

$$AMAT_{512KB} = 4 + (0.05 \times 100) = 9 \text{ cycles} \tag{6}$$

7. (10 pts) Solve problem B.13 on page B-65.

   (a) If a TLB has a hit rate of 98% and a miss penalty of 30 cycles, and each access takes 1 cycle:

$$AMAT_{TLB} = 1 + (0.02 \times 30) = 1.6 \text{ cycles} \tag{7}$$

Homework 3

1. **Data Hazards and Pipeline Timing - Solve Problem C.1 on pages C-71 and C-72.**

```
Loop:   ld      x1, 0(x2)       ; load x1 from address 0 + x2
        addi    x1, x1, 1       ; x1 = x1 + 1
        sd      x1, 0(x2)       ; store x1 at address 0 + x2
        addi    x2, x2, 4       ; x2 = x2 + 4
        sub     x4, x3, x2      ; x4 = x3 - x2
        bnez    x4, Loop        ; branch to Loop if x4 != 0
```

Assume that the initial value of `x3` is `x2+396`

(a) **Data hazards are caused by data dependencies in the code. Whether a dependency causes a hazard depends on the machine implementation (i.e., number of pipeline stages). List all of the data dependecies in the code above. Record the register, source instruction, and destination instruction; for example, there is a data dependency for register `x1` from the `ld` to the `addi`.**

Register	Source Instruction	Destination Instruction	Explanation
`x1`	`ld x1, 0(x2)`	`addi x1, x1, 1`	`addi` uses loaded value
`x1`	`addi x1, x1, 1`	`sd x1, 0(x2)`	`sd` stores updated value
`x2`	`addi x2, x2, 4`	`sub x4, x3, x2`	`sub` uses updated value
`x4`	`sub x4, x3, x2`	`bnez x4, Loop`	`bnez` uses result of `sub`
`x2`	`addi x2, x2, 4`	`ld x1, 0(x2)(next iter.)`	New address for load
`x2`	`addi x2, x2, 4`	`sd x1, 0(x2)(next iter.)`	New address for store

(b) **Show the timing of this instruction seqeunce for the 5-stage RISC pipeline without any forwarding or bypassing hardware but assuming that a register read and a write in the same clock cycle "forwards" through the register file, as between the `add` and `or` shown in Figure C.5. Use a pipeline timing chart like that in Figure C.8. Assume that the branch is handled by flushing the pipeline. If all memory references take 1 cycle, how many cycles does this loop take to execute?**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ld x1, 0(x2)	F	D	X	M	W													
addi x1, x1, 1		F	s	s	D	X	M	W										
sd x1, 0(x2)			F	s	s	D	X	M	W									
addi x2, x2, 4				F	D	X	M	W										
dsub x4, x3, x2					F	s	s	D	X	M	W							
bnez x4, Loop						F	s	s	D	X	M	W						
LD R1, 0(R2)													F	D				

Since the initial value of x3 = x2 + 396 and each iteration of the loop adds 4 to x2, the total number of iterations is 99.

Notice that there are eight cycles lost to RAW hazards including the branch instruction. Two cycles are lost after the branch because of the instruction flushing.

It takes 16 cycles between loop instances so, the total number of cycles is 98 × 16 + 18 = 1586

The last loop takes two additional cycles since this latency cannot be overlapped with additional loop instances.

(c) **Show the timing of this instruction sequence for the 5-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.8. Assume that the branch is handled by predicting it as not taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ld x1, 0(x2)	F	D	X	M	W												
addi x1, x1, #1		F	D	s	X	M	W										
sd x1, 0(x2)			F	s	D	X	M	W									
addi x2, x2, #4				F	D	X	M	W									
dsub x4, x3, x2					F	D	X	M	W								
bnez x4, Loop						F	s	D	X	M	W						
(incorrect instruction)							F	s	s	s	s						
ld x1, 0(x2)												F	D	X	M	W	

Again, we have 99 iterations. There are two RAW stalls and a flush after the branch since the branch is taken. The total number of cycles is 9 × 98 + 12 = 894

The last loop takes three additional cycles since this latency cannot be overlapped with additional loop instances.

(d) **Show the timing of this instruction sequence for the 5-stage RISC peipeline with full forwarding and bypassing hardware, as shown in Figure C.6. Use a pipeline timing chart like that shown in Figure C.8. Assume that the branch is handled by predicting it as taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ld x1, 0(x2)	F	D	X	M	W													
addi x1, x1, #1		F	D	s	X	M	W											
sd x1, 0(x2)			F	s	D	X	M	W										
addi x2, x2, #4				F	D	X	M	W										
dsub x4, x3, x2					F	D	X	M	W									
bnez x4, Loop						F	s	D	X	M	W							
ld x1, 0(x2)												F	D	X	M	W		

Again, we have 99 iterations. We still experience two RAW stalls, but since we correctly predict the branch, we do not need to flush after the branch. Thus, we have only 8 × 98 + 12 = 796 cycles.

(e) **High-performance processors have very deep pipelines - more than 15 stages. Imagine that you have a 10-stage pipeline in which every stage of the 5-stage pipeline has been split in two. The only catch is that, for data forwarding, data are forwarded from the end of a *pair of stages* to the beginning of the two stages where they are needed. For example, data are forwarded from the output of the second execute stage to the input of the first execute stage, still carrying a 1-cycle delay. Show the timing of this instruction sequence for the 10-stage RISC pipeline with full**

forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.8 (but with stages labeled IF1, IF2, ID1, etc.). Assume that the branch is handled by predicting it as taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1
ld x1, 0(x2)	F1	F2	D1	D2	X1	X2	M1	M2	W1	W2						
addi x1, x1, #		F1	F2	D1	D2	s	s	s	X1	X2	M1	M2	W1	W2		
sd x1, 0(x2)			F1	F2	D1	D2	s	s	s	X1	X2	M1	M2	W1	W2	
addi x2, x2, #4				F1	F2	D1	D2	s	s	s	X1	X2	M1	M2	W1	W
dsub x4, x3, x2					F1	s	s	s	F2	D1	D2	s	X1	X2	M1	M
bnez x4, Loop						F1	F2	D1	s	D2	X1	X2	M1	M2	W1	W
ld x1, 0(x2)							F1	F2	s	D1	D2	X1	X2	M1	M2	W

We again have 99 iterations. There are three RAW stalls between the ld and addi, and one RAW stall between the daddi and dsub. Because of the branch prediction, 98 of those iterations overlap significantly. The total number of cycles is 10 × 98 + 19 = 999

(f) **Assume that in the 5-stage pipeline, the longest stage requires** 0.8 ns**, and the pipeline register delay is 0.1** 0.1 ns**. What is the clock cycle time of the 5-stage pipeline? If the 10-stage peipeline splits all stages in half, what is the cycle time of the 10-stage machine?**

Clock-cycle time = Cycle time for longest stage + Register delay

For the 5-stage pipeline: Clock-cycle time = 0.8 + 0.1 = 0.9 ns

For the 10-stage pipeline: Clock-cycle time = $\frac{0.8}{2} + 0.1 = 0.5$ ns

(g) **Using your answers from parts (d) and (e), determine the cycles per instruction (CPI) for the loop on a 5-stage piepline and a 10-stage pipeline. Make sure you count only from when the first instruction reaches the write-back stage to the end. Do not count the start-up of the first instruction. Using the clock cycle time calculated in part (f), calculate the average instruction execute time for each machine.**

We know that: CPI = $\frac{\text{Number of cycles in Total}}{\text{Total Instructions Executed}}$

Average Instruction Time = CPI × Cycle Time

CPI of 5-stage Pipeline = $\frac{796}{99 \times 6} = 1.34$

Avg Inst Exe Time 5-stage = 1.34 × 0.9 = 1.21 ns

CPI of 10-stage Pipeline = $\frac{999}{99 \times 6} = 1.68$

Avg Inst Exe Time 10-stage = 1.68 × 0.5 = 0.84 ns

2. **Branch Hazards - Solve Problem C.2 on page C-72.**

Instruction	15%
Jumps and calls	1%
Taken conditional branches	60% are taken

(a) **We are examining a four-stage pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Assuming that only the first pipe**

stage can always be completed independent of whether the branch is taken and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?

The performance of ideal pipeline without branch hazards is the pipeline depth, which is 4.

Next, for branch hazards, we need to figure out the stall cycles for each type of branches:

- For unconditional branches, the stall cycle is 1.
- For conditional branches which is taken, the stall cycles is 2.
- For conditional branches which is not taken, the stall cycle is 1.

Considering the frequencies of different types of branches:

The performance of pipeline with branch hazards = $\frac{1}{1+\text{pipeline stalls}} \times$ pipeline depth

= $\frac{4}{1+(1\times1\%+2\times15\%\times60\%+1\times15\%\times40\%)} = 3.2$

Speedup = $\frac{\text{Pipeline Speedup without Hazards}}{\text{Pipeline Speedup with Hazards}} = \frac{4}{3.2} = 1.25$

(b) **Now assume a high-performance processor in which we have a 15-deep pipeline where the branch is resolved at the end of the fifth cycle for unconditional branches is resolved at the end of the fifth cycle for unconditional branches and at the end of the tenth cycle for conditional branches. Assuming that only the first pipe stage can always be completed independent of whether the branch is taken and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?**

The stall cycles for different branches are derived as below:

- Unconditional Branches: 4
- Conditional Branches (taken): 9
- Conditional Branches (not taken): 8

The performance of pipeline with branch hazards = $\frac{1}{1+4\times1\%+9\times15\%\times60\%+8\times15\%\times40\%} = \frac{15}{2.33}$

The speedup of the ideal pipeline over the one with branch hazards = $\frac{15}{\frac{15}{2.33}} = 2.33$

3. **Deep Pipeline Performance Analysis - Solve Problem C.7 on page C-75.**

In this problem, we will explre how deepening the pipeline affects performance in two ways: faster clock cycle and increased stalls due to data control and hazards. Assume that the original machine is a 5-stage pipeline with a 1 ns clock cycle. The second machine is a 12-stage pipeline with a 0.6 ns clock cycle. The 5-stage pipeline experiences a stall due to a data hazard every five instructions, whereas the 12-stage pipeline experiences three stalls every eight instructions. In addition, branches constitute 20% of the instructions, and the misprediction rate for both machines is 5%.

(a) **What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account only data hazards?**

Given 5-stage pipeline experiences a stall due to a hazard every 5 instructions and 12 stage pipeline experiences three stalls every eight instructions.

Execution Time = IC × CPI × Cycle Time

Speedup = $\frac{\text{Execution Time of 5-stage Pipeline}}{\text{Execution Time of 12-stage Pipeline}}$

= $\frac{IC\times\frac{6}{5}\times1}{IC\times\frac{11}{8}\times0.6} = 1.45$

(b) **If the branch mispredict penalty for the first machine is 2 cycles but the second machine is 5 cycles, what are the CPIs of each, taking into account the stalls due to branch mispredictions?**

The branch mis predict penalty for the first machine = 2 cycles

The branch mis predict penalty for the second machine = 5 cycles

CPI(5-stage) = Original CPI without branch mispredict + mispredict penalty × percentage of branch instructions × mispredict rate

= $\frac{6}{5} + 0.20 \times 0.05 \times 2 = 1.22$

CPI(12-stage) = $\frac{11}{8} + 0.20 \times 0.05 \times 5 = 1.425$

Speedup = $\frac{IC \times 1.22 \times 1}{IC \times 1.425 \times 0.6} = 1.42$

4. **The following series of branch outcomes occurs for a single branch in a program. (T means the branch is taken, N means the branch is not taken).**

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
Branch Outcome	T	T	N	T	N	T	T	T	T	N	T	T	N

(a) **Assume that we are trying to predict this sequence with a Branch History Table (BHT) using a 1-bit prediction. The counters of the BHT are initialized to the N state. Which of the branches would be mispredicted? Show their indices.**

1-bit predictor results:

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
Outcome	T	T	N	T	N	T	T	T	T	N	T	T	N
Prediction	N	T	T	N	T	N	T	T	T	T	N	T	T
Mispredicted?	Yes	No	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	No	Yes

Total mispredictions: 8 (at indices 1, 3, 4, 5, 6, 10, 11, 13)

(b) **Repeat the above exercise with a 2-bit predictor as shown in Figure C.15 initialized to 10.**

2-bit predictor results (00: strongly not taken, 01: weakly not taken, 10: weakly taken, 11: strongly taken):

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
Outcome	T	T	N	T	N	T	T	T	T	N	T	T	N
State	10	11	11	10	11	10	11	11	11	11	10	11	11
Prediction	T	T	T	T	T	T	T	T	T	T	T	T	T
Mispredicted?	No	No	Yes	No	Yes	No	No	No	No	Yes	No	No	Yes

Total mispredictions: 4 (at indices 3, 5, 10, 13)

5. **Performance Evalution - Solve 3.1 on page 266.**

What is the baseline performance (in cycles, per loop iteration) of the code sequence in Figure 3.47 if no new instruction's execution could be initiated until the previous instruction's execution had completed? Ignore front-end detch and decode. Assume for now that execution does not stall for lack of the next instruction, but only one instruction/cycle can be issued. Assume the branch is taken, and that there is a one-cycle branch delay slot.

The baseline performance (in cycles, per loop iteration) of the code sequence in Figure 3.47, if no new instruction's execution could be initiated until the previous instruction's execution had completed is 37 cycles.

Each instruction requires one clock cycle of execution (a clock cycle in which that instruction, and only that instruction, is occupying the execution units; since every instruction must execute, the loop will take at least that many clock cycles). To that base number, we add the extra latency cycles. Also remember the additional cycle for branch delay slot.

6. **Ideal Dependency Detection - Solve 3.2 on page 267.**

   Think about what latency numbers really mean - they indicate the number of cycles a given function requires to produce its output. If the overall pipeline stalls for the latency cycles of each functional unit, then you are at least guaranteed that any pair of back-to-back instructions (a ”producer” followed by a ”consumer”) will execute correctly. But not all instruction pairs have a producer/consumer relationship. Sometimes two adjacent instructions have nothing to do with each other. How many cycles would the loop body in the code sequence in Figure 3.47 require if the pipeline detected true data dependencies and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? Show the code with `<stall>` cycles to be inserted into the code sequence.) Think of it this way: a one-cycle instruction has latency $1+0$, meaning zero extra wait states. So, latency $1+1$ implies one stall cycle; latency $1+N$ has N extra stall cycles.

   How many cycles would the loop body in the code sequence in Figure 3.47 require if the pipeline detected true data dependencies and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? $\rightarrow$ We need 26 cycles, as shown below.

   Remember, the point of the extra latency cycles is to allow an instruction to complete whatever actions it needs, in order to produce its correct output. Until that output is ready, no dependent instructions can be executed. So, the first fld must stall the next instruction for three clock cycles. The fmul.d produces a result for its successor, and therefore must stall 4 more clocks, and so on.

```
Loop: fld f2, 0(Rx)
      <stall fld>
      <stall fld>
      <stall fld>
      fmul.d f2, f0, f2
      <stall fmul>
      <stall fmul>
      <stall fmul>
      <stall fmul>
      fdiv.d f8, f2, f0
      fld f4, 0(Ry)
      <stall fld>
      <stall fld>
      <stall fld>
      fadd.d f4, f0, f4
      <stall fdiv>
      <stall fdiv>
      <stall fdiv>
      <stall fdiv>
      <stall fdiv>
      fadd.d f10, f8, f2
      fsd f4, 0(Ry)
      addi Rx, Rx, 8
      addi Ry, Ry, 8
      sub x20, x4, Rx
      bnz x20, Loop
```

7. **Dynamic Scheduling - Draw the basic structure of a RISC-V floating point unit for Tomasulo's algorithm. Explain how code is executed with an example.**

Structure of a RISC-V Floating Point Unit for Tomasulo's Algorithm

The Tomasulo's algorithm implementation consists of the following key components:

- **Instruction Queue**: Holds instructions waiting to be issued
- **Reservation Stations**: Hold operations and operands for instructions that have been issued
- **Register File**: Contains the architectural registers
- **Register Status Table**: Tracks which reservation station will produce each register value
- **Functional Units**: Execute the operations (e.g., Load/Store, FP Adders, FP Multipliers)
- **Common Data Bus (CDB)**: Broadcasts results to all waiting reservation stations

Example of a code sequence:

(a) fld f6, 32(x2)
(b) fld f2, 44(x3)
(c) fmul.d f0, f2, f4
(d) fsub.d f8, f2, f6
(e) fdiv.d f0, f0, f6
(f) fadd.d f6, f8, f2

One of the advantages of using Tomasulo's Algorithm is the elimination of WAW and WAR hazards, accomplished by renaming registers using the reservation stations and by storing operands into the reservation station as soon as they are available.

The code sequence issues both the fdiv.d and the fadd.d even though there is a WAR hazard involving f6. The hazard is eliminated in one of two ways. First, if the instruction that provides the value for the fdiv.d has completed, then Vk will store the result, allowing fdiv.d to execute independent of the fadd.d.

On the other hand, if the fld hasn't completed, then Qk will point to the Load1 reservation station, and the fdiv.d instruction will be independent of the fadd.d. Any uses of the result of the fdiv.d will point to the reservation station, allowing the fadd.d to complete and store its value into the registers without affecting the fdiv.d.

The second load has completed effective address calculation but is waiting on the memory unit. The fadd.d instruction, which has a WAR hazard at the WB stage, has issued and could complete before the fdiv.d initiates.

Instruction status			
Instruction	Issue	Execute	Write result
fld f6,32(x2)	✓	✓	✓
fld f2,44(x3)	✓	✓	
fmul.d f0,f2,f4	✓		
fsub.d f8,f2,f6	✓		
fdiv.d f0,f0,f6	✓		
fadd.d f6,f8,f2	✓		

Reservation stations							
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44+Regs[x3]
Add1	Yes	SUB	Mem[32+Regs[x2]]		Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL	Regs[f4]		Load2		
Mult2	Yes	DIV	Mem[32+Regs[x2]]		Mult1		

Register status									
Field	f0	f2	f4	f6	f8	f10	f12	...	f30
Qi	Mult1	Load2		Add2	Add1	Mult2			

Figure 1: Reservation stations and register tags shown when all of the instructions have been issued but only the first load instruction has completed and written its results to the CDB.

Example of a code sequence:

1. fld f6, 32(x2)
2. fld f2, 44(x3)
3. fmul.d f0, f2, f4
4. fsub.d f8, f2, f6
5. fdiv.d f0, f0, f6
6. fadd.d f6, f8, f2

One of the advantages of using Tomasulo's Algorithm is the elimination of WAW and WAR hazards, accomplished by renaming registers using the reservation stations and by storing operands into the reservation station as soon as they are available.

The code sequence issues both the fdiv.d and the fadd.d even though there is a WAR hazard involving f6. The hazard is eliminated in one of two ways. First, if the instruction that provides the value for the fdiv.d has completed, then Vk will store the result, allowing fdiv.d to execute independent of the fadd.d.

On the other hand, if the fld hasn't completed, then Qk will point to the Load1 reservation station, and the fdiv.d instruction will be independent of the fadd.d. Any uses of the result of the fdiv.d will point to the reservation station, allowing the fadd.d to complete and store its value into the registers without affecting the fdiv.d.

Instruction status			
Instruction	Issue	Execute	Write result
fld f6,32(x2)	✓	✓	✓
fld f2,44(x3)	✓	✓	
fmul.d f0,f2,f4	✓		
fsub.d f8,f2,f6	✓		
fdiv.d f0,f0,f6	✓		
fadd.d f6,f8,f2	✓		

Reservation stations							
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44+Regs[x3]
Add1	Yes	SUB	Mem[32+Regs[x2]]		Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL	Regs[f4]		Load2		
Mult2	Yes	DIV	Mem[32+Regs[x2]]		Mult1		

Register status									
Field	f0	f2	f4	f6	f8	f10	f12	...	f30
Qi	Mult1	Load2		Add2	Add1	Mult2			

Figure 2: Reservation stations and register tags shown when all of the instructions have been issued but only the first load instruction has completed and written its results to the CDB.

The second load has completed effective address calculation but is waiting on the memory unit. The fadd.d instruction, which has a WAR hazard at the WB stage, has issued and could complete before the fdiv.d initiates.

Homework 4

1. **What is the basic idea of a vector processor?**
2. **Draw the block diagram of RISC-V Vector architecture. Briefly explain the various blocks in the architecture.**
3. **With an example, explain the basic idea of the following techniques:**
   (a) **Vector chaining**
   (b) **Strip Mining**
   (c) **Vector Stride**
   (d) **Gather and Scatter**
4. **Problem 5.1 parts (a), (b), (c) (Pages 446-447)**
5. **Explain how the following protocols work:**
   (a) **Snooping coherence protocol**
   (b) **Directory based coherence protocol**
6. **Directory Protocol**
   (a) **Using a state-diagram like figure, show how a read miss from local node is handled by the home node and remote node. Briefly explain how it works**
   (b) **Using a state-diagram like figure, show how a write miss from local node is handled by the home node and remote node. Briefly explain how it works**